Zoea

Research note 39

Towards Human-Like Specifications

Edward McDaid & Sarah McDaid
12 May 2025

Zoea uses knowledge about what constitutes human-like code to guide the generation of software and in so doing it also achieves a dramatic reduction in resource requirements. Can we obtain further benefits along similar lines with specifications?

AI is playing an increasingly important role in software development and central to this is its ability to generate code. Most AIs are trained using large numbers of examples of human-originated content. In the case of AIs that code, this normally means countless programs written by people.

Zoea, on the other hand, uses a completely different approach. It does not rely on training, large language models nor even neural networks. Instead, Zoea utilises explicit coding knowledge and while some of this is derived from human software, the key difference is that Zoea is never exposed to a single line of anyone else's code. This makes Zoea ethical - as well as being transparent and immune to hallucination.

Zoea creates human-like code (HLC) - that is, software which has similar characteristics to code produced by human developers. These characteristics, which are derived from human software, include things like what combinations of instructions are used together and how those instructions are assembled to form solutions. On-going research at Zoea continues to refine and extend this set of characteristics.

Software which does not have the characteristics of human code is, of course, non-human-like code (non-HLC). We know that non-HLC definitely exists and indeed it exhibits much greater variety than HLC. As we shall see, this turns out to be a useful thing.

Copyright Zoea Ltd.

Armed with an understanding the characteristics that correspond to HLC, we can easily determine whether any given piece of code is HLC or non-HLC. This ability also makes it possible to generate HLC efficiently.

Now, it may be that HLC is better in some way than non-HLC, and thus, when we generate code, HLC would be more desirable. Many coders have strong opinions on this subject. However, it is currently unknown whether or not this is actually the case. At some point in the future we would like to address that question.

In the mean time, one thing that is certain is that in terms of its frequency of occurrence, HLC is vanishingly rare, when compared with non-HLC. As a result, any randomly produced piece of code is very likely to be non-HLC. Zoea takes advantage of this fact to reduce the effort required to produce an HLC solution for any problem, often by tens of orders of magnitude. Even if this turns out to be the only benefit of HLC, it represents a massive bonus.

Incidentally, other AIs may or may not produce HLC. Just because they were trained on a bottomless bucket of human code doesn't guarantee that much or even any their output will be human-like. Indeed, unless they are simply parroting their training set, most of the code that they produce is very likely to be non-HLC.

HLC may even have a deeper significance. For one thing, the reason why HLC is so rare is not fully understood. Also, we see the same sort of patterns in the way that people use programming languages and in the way that they use natural languages. This could be due to a number of causes. Perhaps, it has something to do with how our brains are wired. Alternatively, it might be a side effect of the incremental process through which we acquire languages. It is clear that there is still much to learn about HLC.

Over the last few years Zoea has systematically explored different aspects of HLC and along the way we have published many of our results. However, there is more to software development than just the production of code. This suggests we also need to examine HLC in a broader context.

Copyright Zoea Ltd.

Zoea automatically generates HLC from a specification, consisting of a set of test cases. Specifications are an important and obvious part of software development. However, they don't get as much research attention as the more technical aspects of coding, such as programming languages. This could be an oversight and if so then it wouldn't be the first time that an apparently fruitless area of study turned out to be anything but.

So maybe we can draw parallels between HLC and specifications. We can easily conceive of a human-like specification (HLS) - that is, a specification that has similar characteristics to one produced by a human. Indeed, there can be no doubt that HLSs exist since most specifications are produced by people. To be useful, there should also be such a thing as non-HLS, which has discernible differences from HLS and which we can take advantage of, in some way. Experience with HLC suggests that this could well be the case.

There are a number of ways in which HLS could turn out to be useful. Most people look at specifications and just see specifications. However, the production of a specification is a key part of software development. Coding utilises many different forms of knowledge and this expertise is imprinted - to some extent - on the artefacts that are produced. Someone with a lot of experience should produce specifications that are different to and (hopefully) better than those produced by a novice. We can potentially identify, extract and leverage whatever knowledge they contain.

Specifications are central to every aspect of how Zoea works. Internally, test cases form the basis for all knowledge representation and reasoning. Zoea comprises a large number of knowledge sources and each of these transforms one set of test cases into another set of test cases, that are different in some way. Most of these test cases are therefore synthetic, in the sense that they have not been produced by people. HLS likely embodies useful coding knowledge - at the specification level - and this has the potential to be widely applied throughout Zoea. For example, being able to identify HLSs would allow us to prioritise knowledge source activations where synthetic test cases are human-like.

When we talk about HLS we recognise that few developers - aside from those using Zoea - currently produce specifications exclusively as comprehensive sets of test cases. Those using traditional programming languages may produce some or even many test cases, but will often define functionality as user stories, use cases or through a number of other techniques. What we mean by HLS in these circumstances is the equivalent specification,

Copyright Zoea Ltd.

if it were articulated entirely as test cases, by a person. Such a specification, if fed into Zoea, would produce a functionally equivalent program. Conversely, non-HLS relates to a similar specification, for a different but functionally equivalent program, which is very likely, but not guaranteed, to be non-HLC.

The nature of the relationship between HLS and HLC - if any - is currently unknown and will have to be determined experimentally. It could take one of several forms. In the simplest scheme, HLS always corresponds to HLC and non-HLS always corresponds to non-HLC. Other schemes are possible, such where non-HLS additionally corresponds to HLC, and so on. It is of course also possible that there is no relationship.

Regardless of the scheme, some general properties of HLS are readily apparent from first principles. For a start, every subdivision of an HLS is also an HLS, while parts of a non-HLS might be either HLS or non-HLS. Intuitively, we know that for any given HLC there are likely to be very many corresponding HLSs. So the size of HLS space will be much larger than HLC space, although non-HLS space will be very much larger still. This suggests that HLS could serve as another very effective heuristic.

A significant challenge in the investigation of HLS is the almost total lack of available human-originated specifications. Unfortunately, there are not yet enough Zoea programs in existence to support such a study. While there is a lot of open source code, the same is not true for specifications. Indeed, few - if any - complete HLS likely even exist in practise. Unit and functional test code are common, but rarely represent specifications that are sufficiently complete to be of any use here. Also, generating sequential or random specifications would not be feasible approaches as these would almost never map to HLC.

One available option is for specifications to be reverse engineered from generated code. In this case we would have to be guided by the scheme as to whether a specification is likely to be HLS based on whether the code is HLC or not. This is only possible for schemes with strong relationships between HLS and HLC. These are also the schemes that we would consider most likely to exist - as well as being the most useful. We can easily produce working HLC and non-HLC programs in any quantity. Fortunately, Zoea also provides existing facilities for test case generation (fuzzing) and test case minimisation. This means that the production of corresponding specifications can be achieved easily and efficiently.

Copyright Zoea Ltd.

Given a sufficiently large quantity of code (HLC / non-HLC) and the corresponding specifications (HLS / non-HLS), it is a fairly straightforward problem to identify what characteristics - if any- separate HLS and non-HLS. Whatever scheme applies, it is expected that the distinction between HLS and non-HLS will be manifest as some combination of relative features derived from test case structure, values and metadata. Whether this is formulated as rules, sets or something else is an open question. If, as seems plausible, there is an HLS scheme where we can reliably distinguish HLS from non-HLS then we can start to quantify the benefits.

Learn more at **zoea.co.uk**

Copyright Zoea Ltd.